EMULEX
An Avago Technologies Company

OneCore™ Storage BSD FC/FCoE I+T CAM Driver Guide

Revision 1.6
August 10, 2015

Connect • Monitor • Manage

Table of Contents

1 Overview 4

1.1 Purpose 4

1.2 Emulex Reference Documents 4

1.3 Browser-Based API Documentation 5

2 Basic Components 6

2.1 Common Front-End Components 7

2.2 CAM SIM (Initiator/Target Back-end Interface) 7

3 Building and Installing 8

3.1 OS Requirements 8

3.2 Building 8

3.3 Installing and Using 8

3.3.1 Module Parameters 8

3.3.1.1 Disabling/Re-enabling Initiator and Target Functionality 10

3.3.2 Loading the Driver 11

3.3.3 Target Installation 11

3.3.3.1 Creating LUNs 11

3.3.3.2 Enabling the CTL Front-End Ports 11

3.3.3.3 Unloading the Driver 12

3.3.4 Initiator Usage 13

3.3.5 Monitoring 13

1 Overview

1.1 Purpose

This document provides an overview and instructions for installing, building, and using the OneCore Storage BSD FC/FCoE I+T CAM driver, also referred to as the BSD ocs_fc_cam driver. The BSD ocs_fc_cam driver is a Fibre Channel (FC)/Fibre Channel over Ethernet (FCoE) reference driver that connects Emulex FC/FCoE adapters to FreeBSD's Common Access Methods (CAM) framework for back-end connectivity to initiators and targets.

The BSD ocs_fc_cam driver supports the latest generation of products based on the Emulex Service Level Interface - 4 (SLI®-4), such as:

- XE100-Series 10Gb/40Gb Ethernet Converged Network I/O controller (IOC) and its adapters:
  - OCe14000-series of 10GbE and 40GbE Converged Network Adapters (CNAs), such as the OCe14102-UX and OCe14102-UM.
- XE201 IOC, combinations of 8 Gigabit Fibre Channel (GFC), 16 GFC, and 10 Gigabit Ethernet (GbE), and its adapters:
  - LPe15004 quad-port Advanced-8 8GFC host bust adapters (HBAs)
  - LPe16000-series of HBAs (such as LPe16000B, LPe16202, and LPe16004)

For a complete list and exact part numbers, contact your Emulex support or sales representative.

1.2 Emulex Reference Documents

Applicable Documents Included in the OneCore Storage SDK Release Package

- *OneCore Storage BSD FC/FCoE I+T CAM Driver Guide* (this document)
- *OneCore Storage Driver Design Manual* – provides the architecture and design details for the common front-end FC/FCoE and iSCSI components of the reference and production drivers. It also provides specific details for the reference drivers that use the RAMD as its target back-end, such as the Linux ocs_fc_ramd driver.
- *OneCore Storage FAQs* – provides answers for frequently asked questions (FAQs) for the OneCore Storage drivers and utilities.
- *OneCore Storage Performance Tuning Application Note* – contains details to improve Emulex adapter performance when using the OneCore Storage Linux drivers in a multi-core CPU environment. It includes descriptions for CPU affinity, CPU frequency scaling performance settings, and OCS driver port to NUMA node mapping. The ocs_config.py script, located in the tools/bin directory, incorporates these settings.

- *OneCore Storage SDK Release Notes* – lists known issues, resolved issues, and technical tips for a particular OneCore Storage SDK release.
- *OneCore Storage Utilities Manual* – describes the usage and commands of the elxocsutil, elxsdkutil, and mgmt utilities. It also includes helpful SDK information in relation to the SLI-4 specifications.

Additional Reference Documents

- *SLI-4 Adapter Management Commands*
- *SLI-4 Architecture Specification*
- *SLI-4 FC and FCoE Command Reference*

1.3 Browser-Based API Documentation

The OneCore Storage reference drivers and elxsdkutil utility include an HTML-based reference manual that is generated from the source code using the Doxygen tool. This documentation includes function APIs, as well as links for navigating through the source code.

To access the BSD ocs_fc_cam driver's HTML-based documentation, you must do the following:

1. Go to the "driver/bsd/ocs_fc_cam" directory.
2. Extract the ocs_fc_cam-doc.tgz file under the "driver/bsd/ocs_fc_cam" directory. This places the *.html files under the "driver/bsd/ocs_fc_cam/doc/html/" directory.
3. Open the "ocs_sdk_documentation.html" file in your preferred browser. This file is located at the same level as the doc, driver, firmware, and tools directories.
4. Under "Browser-based Documentation (HTML files)," click on the BSD FC/FCoE I+T CAM Driver (BSD ocs_fc_cam)'s "HTML files" link. Alternatively, you can open the index.html file located in the driver/bsd/ocs_fc_cam/doc/html/ folder.

2 Basic Components

As shown in Figure 2-1, the BSD ocs_fc_cam driver is composed of the common FreeBSD FC/FCoE front-end components and the CAM SCSI Interface Modules (SIM) back-end interface component.

- FreeBSD FC/FCoE front-end components:
  - PCI
  - OS Abstraction
  - FC/FCoE transport
  - FC/FCoE hardware abstraction layer (HAL)
  - FC/FCoE SLI-4
- Initiator/target back-end interface:
  - CAM SIM

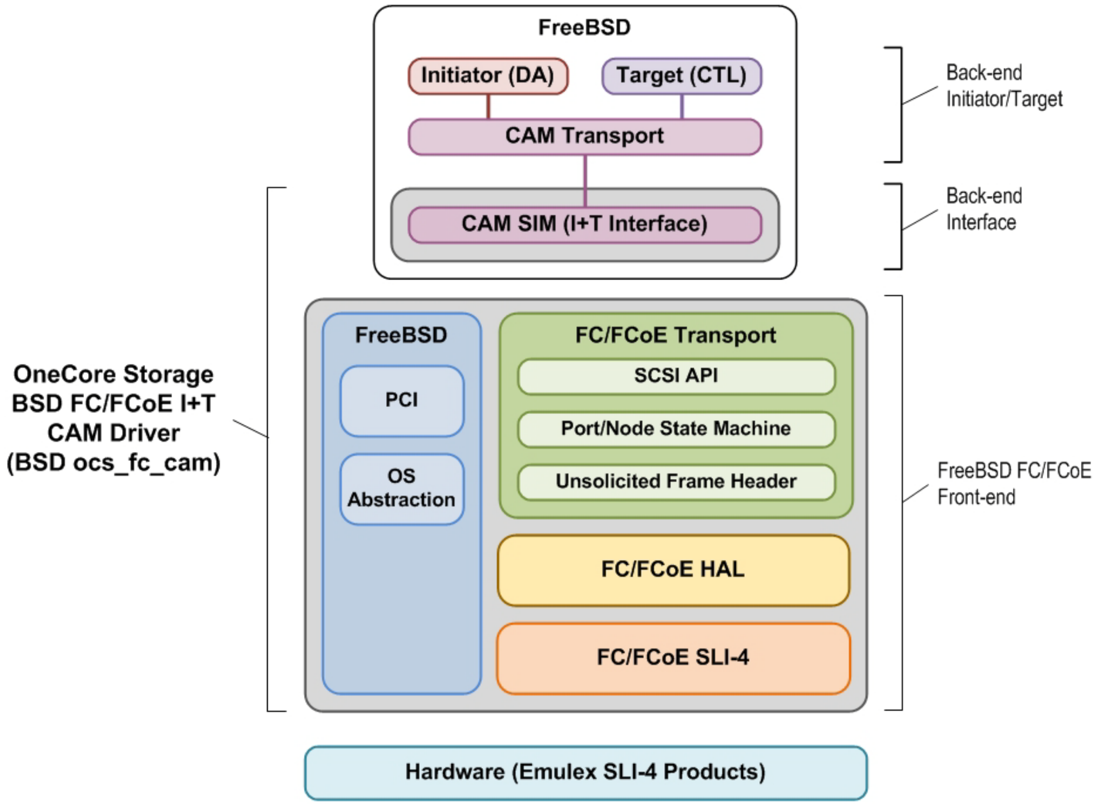

Figure 2-1 BSD ocs_fc_cam driver Framework

2.1 Common Front-End Components

The BSD ocs_fc_cam driver uses the same front-end FC/FCoE components as the other OneCore Storage FC/FCoE reference drivers. Additionally, it uses the same FreeBSD component (PCI and OS abstraction components) as the OneCore Storage BSD FC/FCoE Target RAMD reference driver. For these component details, see the following sections in the latest *OneCore Storage Driver Design Manual*.

Table 2-1 Common FreeBSD and FC/FCoE Front-End Components

Common Front-End Component	*OneCore Storage Driver Design Manual* Section
FC/FCoE SLI-4	“2.1.1 FC/FCoE SLI-4 Component”
FC/FCoE HAL	“2.1.2 FC/FCoE HAL Component”
FC/FCoE Transport	“2.1.3 FC/FCoE Transport Component”
FreeBSD Component	“2.2.1.1 FreeBSD Component”

2.2 CAM SIM (Initiator/Target Back-end Interface)

The CAM SIM, contained in the ocs_cam.c file:

- Registers the front end with FreeBSD’s CAM SCSI subsystem.
- Configures the device for operation.
- Translates device events, requests, or responses into the analogous CAM event, request, or response.

The CAM requires SIM drivers to register device-specific methods including:

- ocs_scsi_tgt_new_device() – Attach the driver to the BSD SCSI layer (that is, CAM).
- ocs_scsi_tgt_del_device() – Tears down target members of the ocs structure.
- ocs_poll() – manually process events.
- ocs_action() – perform the requested action, including
  - Reset bus
  - Path inquiry
  - Get/set transport settings
  - Target operations (enable LUN, notify, accept/continue I/O)
  - Initiator operations (geometry, SCSI I/O, abort/TMF, find targets)

3 Building and Installing

The BSD ocs_fc_cam driver connects Emulex FC/FCoE adapters to FreeBSD's CAM framework for back-end connectivity to initiators and targets. It also works in conjunction with standard FreeBSD tools, such as camcontrol and ctladm.

For additional information on CAM and CTL, see the camcontrol(8), ctladm(8), and stlstat(8) manual pages.

3.1 OS Requirements

For the ocs_fc_cam driver:

- Use FreeBSD 10.1 or later versions.
- Install the latest OS patches for FreeBSD (to enable the latest functionality).

For more information, see the *OneCore Storage SDK Release Notes* (included with the OneCore Storage SDK release package).

3.2 Building

To build the ocs_fc_cam driver:

1. In a FreeBSD system capable of performing kernel builds, extract the files into a work directory.
2. Change to the ocs_fc_cam directory:

   ```
   make depend all
   ```

   This command builds the ocs_fc_cam.ko loadable kernel object module.

3.3 Installing and Using

3.3.1 Module Parameters

The driver supports several load-time parameters available through the following kernel environment variables:

Table 3-1 BSD ocs_fc_cam Module Parameters

Module Parameter	Description
`ddump_saved_size`	Indicates the size, in bytes, of a saved driver dump. Generating a saved driver dump requires additional driver code changes. The size can be any positive number. The default is 0 (disabled).

Table 3-1 BSD ocs_fc_cam Module Parameters

Module Parameter	Description
`enable_hlm`	Enables high login mode (HLM) when set to 1. The default is 0 (disabled). When HLM is disabled, then one remote port indicator (RPI) per login is allocated. Enabling HLM may be necessary when an application may exceed the SLI-4/adapter RPI constraints. When HLM is enabled, the number of logins is limited only by driver resources (memory). When HLM is enabled, remote connections that share the same service parameters can share an RPI. After an RPI and service parameters are registered with the adapter, subsequent requests (target send/receive/response requests, initiator send command requests) will specify the destination FC_ID in the request. If enabled, then RPIs are allocated as follows: • A unique RPI is allocated for each of the first hlm_group_size logins from remote nodes with compatible service parameters. • Subsequent login requests in excess of the hlm_group_size will use (share) one of the previously allocated RPIs on a round robin basis.
`explicit_buffer_list`	Host dynamically assigns SGLs as needed when set to 1. Driver pre-registers fixed-value SGLs when set to 0 (default).
`external_loopback`	Enables external loopback mode when set to 1. This mode allows a port to discover and log in to itself when the Tx and Rx signals are connected through a loopback hood. The default is 0 (external loopback mode disabled).
`hlm_group_size`	Sets the high login mode group size. Possible values are 1–8192. The default is 8. The low default value is chosen to facilitate testing of the feature; in a typical scenario, it is expected that a higher value would be specified.
`initiator`	Enables initiator functionality. Default is 1 (enabled). Set to 0 to disable.
`loglevel`	Indicates the type and amount of log output. Each successive loglevel includes the information of the lower levels. For example, loglevel=3 (default) includes LOG_INFO, LOG_WARN, LOG_ERR, and LOG_CRIT information. Valid values are 0 - 5: • 0 - LOG_CRIT: Logs critical information. The hardware is not usable. The chip is unresponsive or in a Unrecoverable Error (UE) state. • 1 - LOG_ERR: Logs error information that may prevent normal operation, such as out-of-memory or start-up failures. • 2 - LOG_WARN: Logs warnings and error information. The driver is functional, but the requested operation may not have worked as expected (for example, limits may have been exceeded or I/O errors may have occurred). • 3 - LOG_INFO: Logs informational messages. Operation is normal, but something of interest may have occurred, such as a driver start or link up/down. • 4 - LOG_TEST: Logs test information. Operation is normal, but an error may have occurred and has been handled successfully. • 5 - LOG_DEBUG: Logs debugging information for development and test. Operation is normal. The default is “3”, LOG_INFO, if no number is selected.

Table 3-1 BSD ocs_fc_cam Module Parameters

Module Parameter	Description
`logmask`	Mask bits to enable logging: • bit[0] - Set to 1 to enable node state machine traces. • bit[1] - Set to 1 to enable ELS traces. • bit[2] - Set to 1 to enable SCSI command traces. • bit[3] - Set to 1 to enable SCSI back-end target traces. • bit[4] - Set to 1 to enable domain and sport state machine traces. • bit[5] - Set to 1 to enable the logging of messages with a status of SCSI Task Set Full (Queue Full) or Busy. The default value for each of the bits is 0 (that is, logging is disabled by default). Mask values are also defined in ocs_debug.h.
`num_vports`	Specifies the number of NPIV ports to create. The default is 0. The NPIV port WWNN is a type 2 address derived from a factory-supplied MAC address. The NPIV WWPN is derived by OR'ing the NPIV port ID (also known as VPI) into bits 59:48 of the WWNN.
`speed`	Sets the link speed. Value is in megabits per second. Possible values include: • 0 – Auto-speed negotiation (default) • 4000 (4GFC) • 8000 (8GFC) • 10000 (10GbE) • 16000 (16GFC)
`target`	Enables target functionality. Default is 1 (enabled). Set to 0 to disable.
`target_io_timer`	**Note:** This parameter is only applicable when the target parameter is 1. The timeout value, in seconds, for target commands. The default is 0 (target_io_timer disabled).
`topology`	Specifies FC topology: • 0 – auto-select; first tries N_Port, then loop. (default) • 1 – N_Port (point-to-point) • 2 – loop

3.3.1.1 Disabling/Re-enabling Initiator and Target Functionality

For the ocs_fc_cam driver, both target and initiator functionality are enabled by default. You can disable their functionality by using the following commands.

- To disable the target functionality on port 0 and port 1:

```
kenv hint.ocs_fc_cam.0.target=0
kenv hint.ocs_fc_cam.1.target=0
```

- To disable the initiator functionality on port 0 and port 1:

```
kenv hint.ocs_fc_cam.0.initiator=0
kenv hint.ocs_fc_cam.1.initiator=0
```

For the change to take effect, you must unload and then re-load the driver.

If you disable functionality and want to re-enable it, use the "-u" option to delete the variable(s). For example, if you disabled target functionality for port 0 and want to re-enable it, use the following command:

```
kenv -u hint.ocs_fc_cam.0.target
```

For the change to take effect, you must unload and then re-load the driver.

3.3.2 Loading the Driver

To load the driver, as root:

```
kldload ./ocs_fc_cam.ko
```

Note: This kldload command must be executed in the same directory as the ocs_fc_cam.ko file, otherwise, you must include the complete path to the ocs_fc_cam.ko file.

3.3.3 Target Installation

3.3.3.1 Creating LUNs

After loading the driver, the user must create LUN(s) with ctladm. The CTL supports two types of LUNs: ramdisk and block. The ramdisk LUNs can be used for performance measurements, but they do not provide data integrity. Block LUNs provide data integrity, as they use either a file or block device for their backing storage.

Examples for creating ramdisk and block LUNs:

```
ctladm create -b ramdisk -s 1099511627776
ctladm create -b block -o file=/target_file00.raw
```

One method for creating a file backing store is to use truncate:

```
truncate -s 1G /target_file00.raw
```

Note: It is possible that sporadic issues may occur if the file backing store lives on a UFS file system. Better results have been observed if the file backing store lives on a ZFS file system, or using a disk device. For example:

```
ctladm create -b block -o file=/dev/ada1
```

3.3.3.2 Enabling the CTL Front-End Ports

Note: An issue appears to exist with the SYNCHRONIZE CACHE commands and GEOM. To avoid this issue, configure the CTL from forwarding synchronize commands to the back-end by issuing the following command:

```
ctladm realsync off
```

This must be done prior to enabling the front-end ports (that is, before issuing "ctladm port -o on").

After creating the LUNs, you must enable the corresponding front-end ports.

1. Display front-end port status:

   ```
   ctladm port –l
   ```

   This output shows that the target ports (ocs_fc_cam0 and ocs_fc_cam1) are not online:

   ```
   Port Online Type     Name         pp vp WWNN               WWPN
   0    NO     IOCTL    CTL ioctl    0  0  0                  0
   1    NO     INTERNAL ctl2cam      0  0  0x5000000f27aa0300 0x5000000f27aa0302
   2    NO     INTERNAL CTL internal 0  0  0                  0
   3    NO     FC       ocs_fc_cam0  0  0  0x20000000c9d1a468 0x10000000c9d1a468
   4    NO     FC       ocs_fc_cam1  1  0  0x20000000c9d1a469 0x10000000c9d1a469
   ```

2. Enable the front-end ports:

   ```
   ctladm port -o on -t fc
   ```

3. To confirm that the front-end ports are online, display the front-end port status:

   ```
   ctladm port –l
   ```

   This output shows the target ports (ocs_fc_cam0 and ocs_fc_cam1) being online:

   ```
   Port Online Type      Name          pp   vp   WWNN                WWPN
   0    NO     IOCTL     CTL ioctl     0    0    0                   0
   1    NO     INTERNAL  ctl2cam       0    0    0x5000000f27aa0300  0x5000000f27aa0302
   2    NO     INTERNAL  CTL internal  0    0    0                   0
   3    YES    FC        ocs_fc_cam0   0    0    0x20000000c9d1a468  0x10000000c9d1a468
   4    YES    FC        ocs_fc_cam1   1    0    0x20000000c9d1a469  0x10000000c9d1a469
   ```

3.3.3.3 Unloading the Driver

Before unloading the driver, you must disable the corresponding CTL front-end ports and remove the LUNs.

1. Disable the corresponding CTL front-end ports:

   ```
   ctladm port -o off -t fc
   ```

   Output:

   ```
   Front End ports disabled
   ```

2. Remove LUN 0 of a previously created ramdisk:

   ```
   ctladm remove -b ramdisk -l 0
   ```

   Output:

   ```
   LUN 0 deleted successfully
   ```

3. Unload the driver:

   ```
   kldunload ocs_fc_cam
   ```

3.3.4 Initiator Usage

After loading the driver, run the following command to display a list of available devices:

```
camcontrol devlist
```

The output of the command should look similar to the following:

```
<ST3160318AS CC45>at scbus0 target 1 lun 0 (pass0, ada0)
<TEAC DV-18S-A B.0C>at scbus3 target 0 lun 0 (cd0,pass1)
<FREEBSD CTLDISK 0001>at scbus4 target 1 lun 0 (pass2)
<FREEBSD CTLDISK 0001>at scbus5 target 1 lun 0 (pass3,da0)
<FREEBSD CTLDISK 0001>at scbus5 target 1 lun 1 (pass4,da1)
<FREEBSD CTLDISK 0001>at scbus5 target 1 lun 2 (pass5,da2)
<FREEBSD CTLDISK 0001>at scbus5 target 1 lun 3 (pass6,da3)
<FREEBSD CTLDISK 0001>at scbus6 target 1 lun 0 (pass7,da4)
<FREEBSD CTLDISK 0001>at scbua6 target 1 lun 1 (pass8,da5)
<FREEBSD CTLDISK 0001>at scbus6 target 1 lun 2 (pass9,da6)
```

To run I/O LUNs da4 and da5:

```
fio --output=fio_test_run.txt –rw=rw --bs=512 --iodepth=8 --direct=1
--runtime=120 --time_based --name=/dev/da4 --name=/dev/da5
```

This command writes the output to the fio_test_run.txt file, runs mixed sequential reads and writes with a block size of 512 bytes, and maintains an I/O depth (queue depth) of 8 for 120 seconds to LUNs /dev/da4 and /dev/da5. For more information, see the fio(1) man page.

3.3.5 Monitoring

To monitor I/O performance and CPU utilization, use the ctlstat command.